Simulación Educativa Mediante Meta-Modelado y Gramáticas de Grafos

Juan de Lara Jaramillo
Escuela Politécnica Superior
Ingeniería Informática
Universidad Autónoma de Madrid
e-mail: Juan.Lara@ii.uam.es

Resumen

En este artículo se propone el uso de meta-modelado y de gramáticas de grafos como un medio de especificar formalmente simulaciones discretas con un enfoque educativo. Por medio del meta-modelado, se pueden describir de forma gráfica formalismos de simulación y utilizar los meta-modelos para generar automaticamente herramientas de modelado para estos formalismos. Por medio de las gramáticas de grafos, se pueden describir de manera formal el tipo de manipulaciones permitidas sobre los modelos (optimizaciones, transformaciones, simulaciones, generación de código, etc.). En particular en este artículo, se presenta un ejemplo en el que se describe un lenguaje gráfico sencillo de simulación discreta al estilo de *Interacción de Procesos*. Se presenta además una gramática de grafos que permite la simulación de los modelos definidos mediante este formalismo, de tal forma que se puede observar paso a paso de forma gráfica la evolución de la simulación.

Palabras Clave: Meta-Modelado, Gramáticas de Grafos, Simulación Discreta, Interacción de Procesos

1. Introducción

La simulación es una de las áreas más antíguas de la informática y se ha utilizado frecuentemente con un enfoque educativo, a veces (sobre todo si éstas son interactivas), en situaciones en las que se quiere que el estudiante desarrolle un papel activo en el aprendizaje. Esto se debe a que las simulaciones interactivas permiten que el estudiante estudie el comportamiento de un sistema mediante la experimentación (cambiando valores iniciales de parámetros, la estructura del modelo, etc.), respondiendo a preguntas del tipo "*...qué pasaría si...?*".

El meta-modelado consiste en modelar los propios formalismos mediante la construcción de meta-modelos. Las herramientas de meta-modelado [deLa02] permiten generar otras herramientas de modelado para el formalismo descrito, a partir de la información de los meta-modelos. Esto permite construir entornos de modelado gráfico con gran rapidez, ya que los formalismos que se utilizan para el meta-modelado suelen ser gráficos (como por ejemplo los diagramas de clases UML). Las gramáticas de grafos son un formalismo gráfico para la especificación formal de transformaciones en grafos. Como los (meta-)* modelos se pueden almacenar como grafos, las gramáticas de grafos resultan un mecanismo natural para la especificación de computaciones en modelos, que pueden complementar las capacidades de las herramientas generadas mediante meta-modelado.

En este artículo se presenta una herramienta (AToM^3 [ATOM03] [deLa02]) que implementa los conceptos expuestos anteriormente. La herramienta permite definir formalismos (gráficamente) mediante meta-modelado, y expresar manipulaciones de modelos (gráficamente) mediante gramáticas de grafos. Además se presenta un ejemplo en el que se define la sintaxis de un lenguaje visual de simulación discreta, así como su semántica operacional (simulador) mediante gramáticas de grafos. Se ha seguido un enfoque educativo, ya que el alumno tiene acceso a la especifiación de las reglas del simulador. Estas, al ser gráficas, permiten una fácil comprensión del funcionamiento de la simulación por parte del estudiante. Además, la simulación

propiamente dicha se realiza de forma visual, siendo posible la visualización de cualquier parámetro del modelo, así como su modificación en tiempo de ejecución.

El artículo se ha organizado como sigue: la sección 2 da una breve introducción a las gramáticas de grafos. La sección 3 presenta algunos conceptos básicos de meta-modelado. En la sección 4 se explica brevemente la herramienta AToM^3. En la sección 5, se describe el formalismo de simulación discreta propuesto (mediante meta-modelado, con AToM^3), así como su semántica operacional (mediante gramáticas de grafos, también con AToM^3). La sección 6 presenta herramientas y trabajos similares, y finalmente la sección 7 discute las conclusiones y el trabajo futuro.

2. Gramáticas de Grafos

En el enfoque que se presenta, los (meta-)*modelos se almacenan como grafos, y se usan gramáticas de grafos para expresar manipulación de estos modelos. Algunas de las manipulaciones que resultan interesantes en el campo del modelado y la simulación incluyen:

- Transformación de un modelo expresado en un formalismo a otro modelo (equivalente en cuanto al comportamiento) pero expresado en otro formalismo distinto [deLa02b]. Esto es útil si el formalismo destino permite la respuesta a preguntas que el formalismo origen no permitía.
- Optimización de modelos, por ejemplo, reduciendo su complejidad.
- Simulación de modelos, es decir, expresar la semántica operacional de los formalismos (en este artículo se presenta un ejemplo de este tipo de manipulaciones).
- Generación de código (textual) para su procesamiento posterior mediante otras herramientas.

En analogía con las gramáticas de Chomsky sobre cadenas, las gramáticas de grafos [Ehri99] se pueden usar para describir transformaciones en grafos, o para generar conjuntos de grafos válidos. Las gramáticas de grafos están compuestas por reglas, y éstas por partes izquierdas y derechas (pre- y post- condiciones) en las que aparecen grafos. Las gramáticas de grafos se aplican a un grafo de entrada, para transformarlo. Cuando se encuentra una correspondencia (homomorfismo) entre la parte izquiera de una regla y un subgrafo del grafo de entrada, se reemplaza este subgrafo por la parte derecha de la regla. Las reglas pueden contener también una condición, que debe ser satisfecha para que se pueda aplicar la regla, así como acciones que se ejecutan si la regla es aplicada. Los sistemas de reescritura de grafos aplican iterativamente las reglas de la gramática al grafo, hasta que ninguna regla es aplicable [Dörr95]. Algunos enfoques ofrecen también especificaciones para el control de flujo en la ejecución de las reglas. En AToM^3, se asigna una prioridad a las reglas y se ordenan de acuerdo a ella. Cuando una regla se aplica, el sistema de reescritura comienza a comprobar la aplicabilidad de las reglas de nuevo por el principio de la lista.

Por una parte, el uso de un modelo (en forma de gramáticas de grafo) para representar manipulaciones de modelos tiene ciertas ventajas sobre una representación implícita (es decir, representando la computación que realiza la transformación en un programa textual) [Blon96]. Por ejemplo, las gramáticas de grafos son una representación gráfica, abstracta, formal, declarativa y de alto nivel de las computaciones. Además, los fundamentos teóricos de los sistemas de reescritura de grafos pueden ayudar a probar propiedades de la transformación, como la corrección y la convergencia (terminación).

Por otra parte, el uso de gramáticas de grafos está restringido por la eficiencia. En el caso más general, la comprobación de isomorfismos en grafos es un problema NP-completo, si bien el uso de grafos pequeños en las partes izquierda y derecha de las reglas, así como el uso de tipos y etiquetas en nodos y enlaces reducen mucho el espacio de búsqueda. Esta es la situación de la mayoría de los formalismos de simulación para los que hemos definido gramáticas de grafos.

Para los objetivos del presente trabajo, las ventajas son mayores que los incovenientes. Pese a que se ha implementado un simulador mediante gramáticas de grafos, no se ha pretendido que éste sea extremadamente eficiente. La ventaja es que el simulador se ha podido implementar de forma totalmente gráfica (a la par que rigurosa), lo cual lo hace fácilmente entendible. La ejecución iterada de las reglas de la gramática del simulador resulta en la animación de la simulación.

3. Meta-Modelado

El Meta-modelado consiste en modelar los propios formalismos. Estos se describen como modelos utilizando *meta-formalismos*, que no son más que formalismos lo suficientemente expresivos para expresar la sintaxis (posiblemente gráfica) de otros formalismos. Ejemplos de meta-formalismos son los diagramas Entidad-Relación (DER) o los diagramas de clases UML [Booc99]. Las herramientas de meta-modelado [deLa02] permiten generar de forma automática otras herramientas para modelar en el formalismo descrito por el meta-modelo.

Un modelo de un meta-formalismo se llama meta-meta-modelo, un modelo de un formalismo de llama un meta-modelo. La siguiente tabla muestra los niveles considerados en nuestro enfoque. Sólo consideramos tres niveles, aunque puede ser el caso de que un meta-formalismo MF_1 sea lo suficientemente expresivo como para describir el meta-meta-modelo de otro meta-formalismo MF_2. Consideramos a ambos MF_1 y MF_2 como meta-formalismos y los ponemos en el mismo nivel. Como se verá más tarde, en AToM^3 se da el caso de que meta-formalismos puedan describir formalismos y meta-formalismos.

Nivel	Descripción	Ejemplo
Meta-meta-modelo	Modelo que describe un meta-formalismo. Especificado con un meta-formalismo.	Modelos de ERD, o diagramas de clases UML.
Meta-modelo	Modelo que describe un formalismo de simulación. Especificado con un meta-formalismo.	Descripción de formalismos como Autómatas Finitos, Ecuaciones Diferenciales Ordinarias (ODE), etc
Modelo	Descripción de un objeto utilizando un formalismo.	f'(x) = -sin(x) f(0) =0 (en el formalismo ODE)

Tabla 1: Niveles de meta-modelado

Pero para ser capaces de especificar los formalismos de manera completa, los meta-formalismos pueden tener que ser extendidos con la habilidad de expresar restricciones, para limitar el número de modelos que se consideran correctos. Por ejemplo, cuando se modela el formalismo "*Autómatas Finitos Deterministas*", las transiciones que salen del mismo estado deben tener etiquetas distintas. Esta restricción no se puede expresar utilizando solamente DER o diagramas de clases UML. Mientras que los meta-formalismos utilizan frecuentemente una notación gráfica, las restricciones se pueden expresar de forma concisa añadiendo un lenguaje textual al meta-formalismo. De esta forma, algunos sistemas [Gray00] incluido AToM^3 usan el lenguaje OCL que se usa en el UML. Como AToM^3 [deLa02] se ha implementado utilizando el lenguaje Python [Pyth02] se pueden utilizar expresiones en Python para expresar las restricciones.

Otras alternativas al uso de un lenguaje de restricciones es expresar en una gramática de grafos el tipo de acciones de edición que el usuario puede realizar en cada momento de la fase de modelado. Este enfoque se llama "*dirigido por la sintaxis*" [Bard02]. Otros editores se llaman "*free-hand*" [Mina02] y permiten al usuario más flexibilidad en la fase de edición, pero estos editores deben comprobar que el modelo que el usuario está construyendo es correcto (probablemente mediante un lenguaje de restricciones). En AToM^3, la edición de forma "*free-hand*" es el enfoque por defecto, y la corrección de los modelos se garantiza evaluando las restricciones definidas en el meta-modelo (y asociadas con eventos) cuando el usuario está construyendo el modelo. En AToM^3, la edición de tipo "*free-hand*" se puede combinar con el enfoque "*dirigido por la sintaxis*" mediante la construcción de gramáticas de grafos que describen las acciones de edición.

4. AToM^3

AToM^3 [ATOM03] es una herramienta para el modelado multi-paradigma (que incluye meta-modelado, modelado multi-formalismo y en múltiples niveles de abstracción) que el autor desarrolló en colaboración con el laboratorio MSDL de la Universidad de McGill en Montreal. La idea principal de la herramienta es: "*todo es un modelo*", en el sentido de que incluso la interfaz de usuario de AToM^3 es un modelo, que se interpreta al cargarse, y que por tanto puede modificarse. La estructura de AToM^3 se muestra en la Figura 1.

AToM^3 se puede utilizar como una herramienta de meta-modelado para definir formalismos. Para ello, se tiene disponible el meta-formalismo DER (Diagramas Entidad Relación), extendido con restricciones. Este es el (meta-)formalismo que se carga en AToM^3 por defecto. Utilizando DER, el usuario puede definir el meta-modelo del formalismo que desee definir, simplemente dibujando el diagrama entidad-relación correspondiente. En la Figura 1, puede verse cómo se ha utilizado el meta-formalismo DER para definir el formalismo "*Autómatas*". Una vez que se ha definido el meta-modelo de un formalismo, éste puede cargarse de nuevo en AToM^3 (bootstrapping), de tal forma que el comportamiento de la herramienta cambiará de acuerdo a la sintaxis definida en el meta-modelo. En la Figura 1 puede verse cómo se ha utilizado el formalismo "*Autómatas*" para modelar un autómata particular, el que describe los binarios pares.

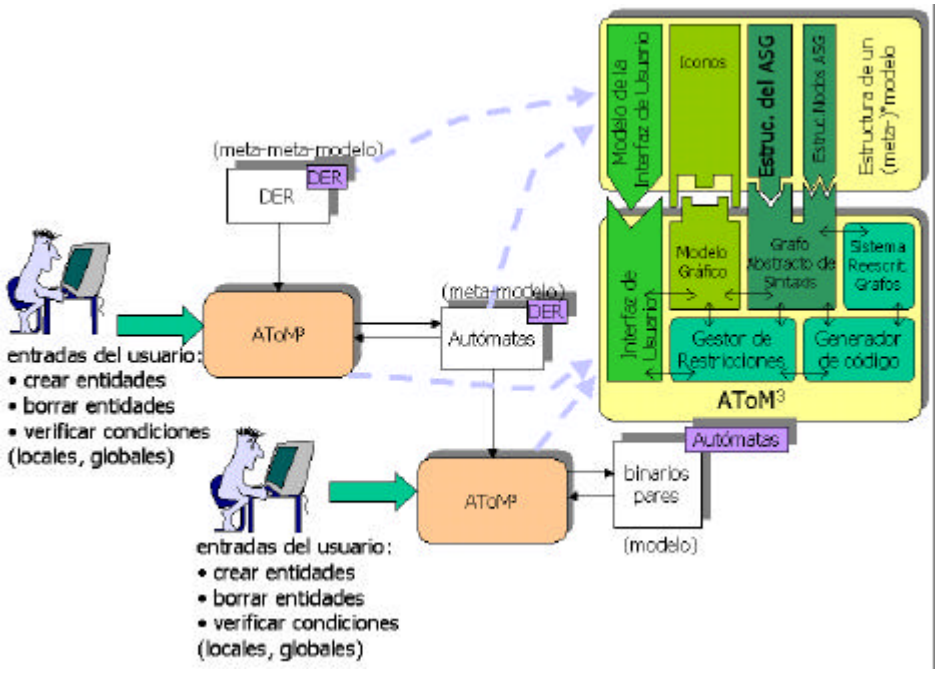

Figura 1: Estructura de AToM^3.

Pero el meta-modelo del formalismo definido mediante meta-modelado sólo nos permite crear modelos sintácticamente correctos, salvarlos y recuperarlos. En el campo del modelado y simulación estamos interesados en manipular estos modelos. Esta manipulación se puede expresar en AToM^3 mediante gramáticas de grafos, que a su vez también son modelos. Para definir una gramática de grafos, se ha de cargar el formalismo "*gramáticas de grafos*", junto con el formalismo para el que se desea definir la transformación ("*Autómatas*", "*Interacción de Procesos*", etc.) Una vez definida la gramática es posible aplicarla sobre modelos definidos con el formalismo, para su transformación.

El lector interesado puede encontrar más información sobre la herramienta en [deLa02], [deLa02b] y en la página web de AToM^3 [ATOM03]. La siguiente sección muestra un ejemplo de las capacidades de la herramienta, en particula su aplicación a la simulación (discreta) educativa.

5. Descripción de un Lenguaje de Simulación Discreta con AToM^3

En esta sección, se va a describir un formalismo gráfico muy sencillo para la simulación discreta al estilo "*Interacción de Procesos*" [Gord96]. La idea de esta visión de la simulación discreta es modelar el proceso que sufre cada una de las transacciones (piezas, personas, etc) que pasa por el sistema (planta de fabricación, supermercado, etc.) El ejemplo típico de lenguaje de simulación discreta del estilo "*Interacción de Procesos*" es GPSS [Gord96]. Este lenguaje ofrece unos 20 bloques (el número exacto depende de la implementación del

compilador) que pueden interconectarse para formar un modelo de simulación. Estos bloques ofrecen funcionalidades como colas, generadores de transacciones, procesos (servidores, máquinas, etc), etc.

En este ejemplo, se va a describir el meta-modelo de un lenguaje gráfico muy sencillo en el que sólo se consideran los bloques generadores, colas y procesos. El meta-modelo del formalismo, descrito mediante DER en AToM^3 se muestra en la Figura 2.

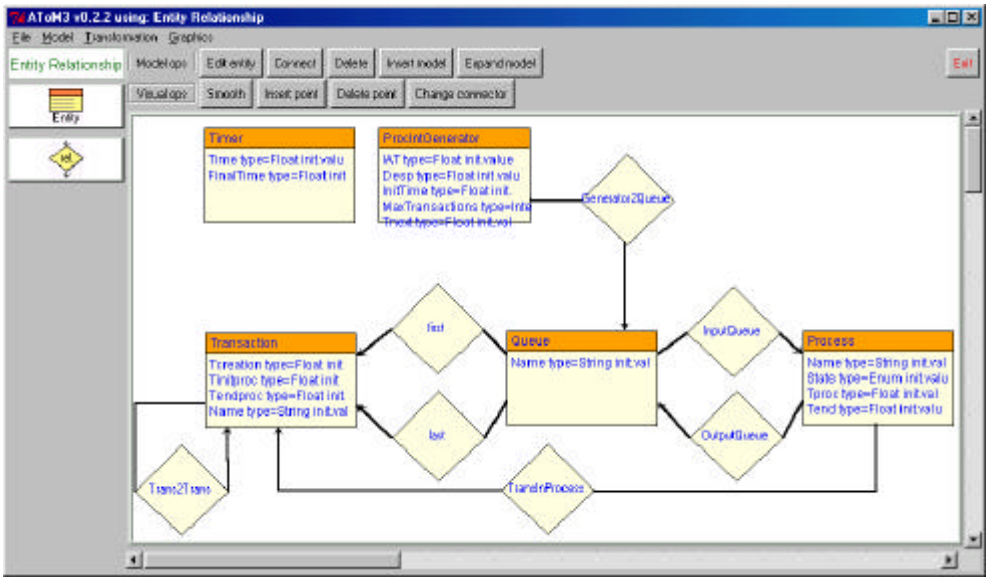

Figura 2: Meta-Modelo del Lenguaje de Simulación del Ejemplo

Como puede observarse, se han definido tres bloques: colas (*Queue*), generadores de transacciones (*ProcIntGenerator*) y procesos (*Process*). Estos últimos pueden estar en estado (atributo *State*) ocioso (*Idle*) u ocupado (*Busy*) y tardan un cierto tiempo (*Tproc*) en completar su tarea. Los generadores son muy parecidos a los que define GPSS, ya que constan de un tiempo medio de llegada (IAT) y un desplazamiento (Desp), de tal forma que la próxima transacción se generará dentro de un número aleatorio de unidades de tiempo en el intervalo *IAT–Desp*. El generador empezará a emitir las transacciones a partir del tiempo *initTime*, y emitirá un máximo de *MaxTransactions*, si éste atributo tiene un valor mayor que cero. Nótese que los generadores sólo pueden conectarse a las colas, es decir generan transacciones que son almacenadas en las colas. Además, en el meta-modelo es necesario definir las entidades transacción (*Transaction*) y una entidad que refleje el tiempo actual de la simulación y el tiempo final (*Timer*). Nótese también que las transacciones pueden estar bien en las colas, o bien en un proceso (cuando éste está en estado "*busy*") y pueden estar relacionadas con otras transacciones que se encuentren en el mismo bloque. Además de estos atributos, a cada una de las entidades y relaciones del meta-modelo se les ha asignado una apariencia gráfica.

A partir del meta-modelo de la Figura 2, AToM^3 genera una serie de ficheros, que al ser cargados en la herramienta, hacen que podamos construir modelos que se ajusten a la sintaxis del meta-modelo. Un ejemplo de uso de AToM^3, cargado con la información del meta-modelo anterior se muestra en la Figura 3. Esta figura muestra un modelo muy sencillo de una planta de fabricación. El modelo está compuesto por un generador, que produce piezas que pueden ser consumidas por dos máquinas (M1 y M2). Como ambas pueden estar ocupadas, son almacenadas en una cola de entrada, compartida por ambas máquinas. Cuando una pieza es procesada por M1 o bien por M2, llega a M3. Antes de ser procesada por M3, puede tener que esperar en una cola de entrada (etiquetada como *conveyor*). Finalmente las piezas terminadas se almacenan en una cola de salida (etiquetada como *Output*). Se ha incluido una entidad *Timer*, que tiene información del tiempo actual y final (50) de la simulación. Así pues la simulación terminará bien cuando el tiempo actual supere las 50 unidades de tiempo, o el generador haya generado 30 piezas (véase el parámetro MAX del bloque generador). Además se han representado también unas cuantas piezas (transacciones), una en la cola de entrada y otras tres siendo procesadas por las máquinas.

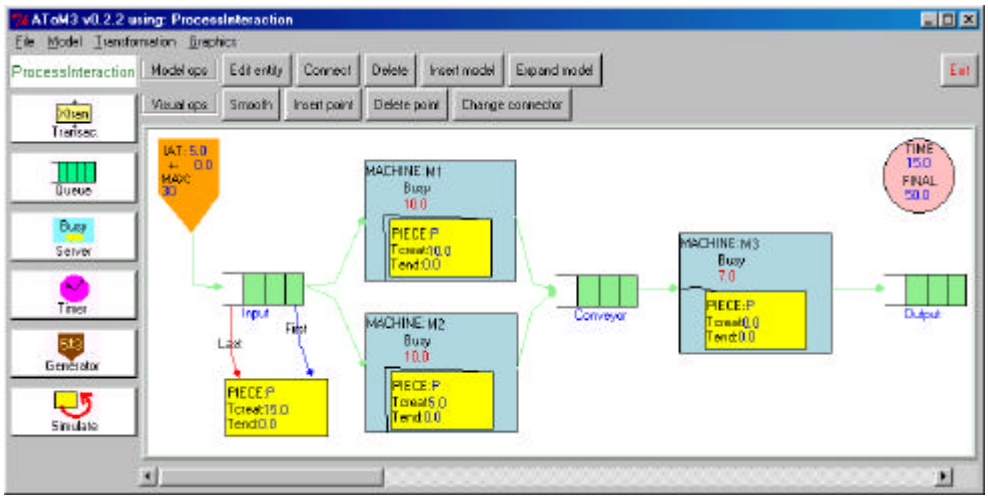

Figura 3: Un Modelo Realizado con el Formalismo de Simulación Definido en el Ejemplo.

Para la simulación de los modelos definidos mediante este formalismo, se ha creado una gramática de grafos que expresa la semántica operacional de este tipo de modelos. En AToM^3, se pueden definir gramáticas de grafos de manera visual. En la Figura 4 se muestran las dos primeras reglas de la gramática.

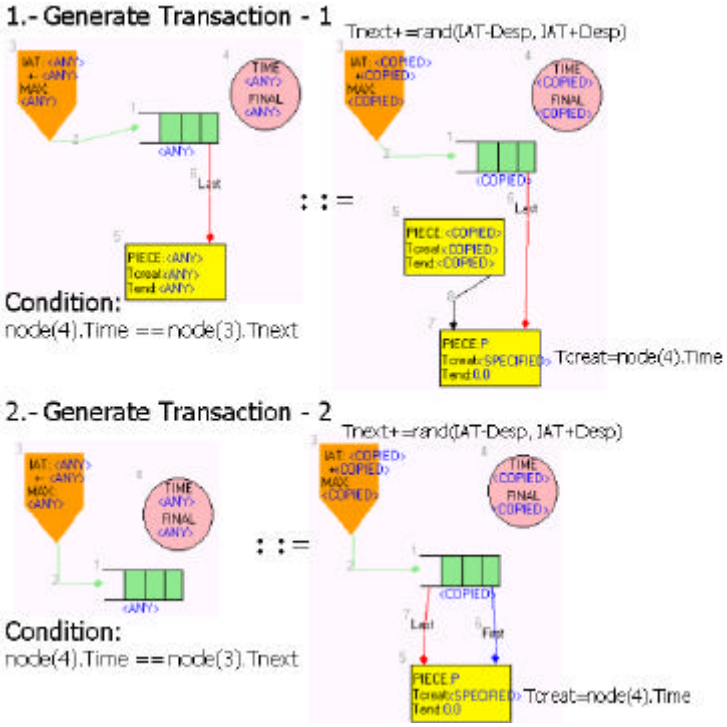

Figura 4: Reglas 1 y 2 de la Gramática para la Simulación: Generación de Transacciones.

La primera regla se puede aplicar si se encuentra el patrón de la parte izquierda en el modelo. Es decir, si en el modelo hay un generador conectado a una cola no vacía (véase la Figura 3 y nótese que en ese modelo se encuentra una ocurrencia del patrón de la regla 1, el grafo formado por el generador, el *timer*, la cola *Input* y la *Pieza P)*. La condición de aplicabilidad especifica que el tiempo actual de la simulación debe ser igual al atributo *Tnext* del generador. En este atributo se almacena el instante de tiempo en el que el generador

producirá la próxima transacción. Si se encuentra una correspondencia entre esta parte izquierda y una zona del modelo, y se cumple la condición de aplicabilidad, entonces podemos reemplazar la zona del modelo por la parte derecha. En la parte derecha de la regla, se ha especificado que se inserta una nueva transacción en la cola, que pasa a ser la última transacción de la cola. Además se actualiza el atributo *Tnext* del generador, así como el campo *Tcreat* de la transacción, que almacena el instante en que ésta se ha creado.

Cabe observar que los nodos y arcos de las partes izquierdas y derechas de las reglas están etiquetados mediante números. Si estos números aparecen en ambas partes, quiere decir que el nodo o arco se mantiene cuando se aplica la regla. Si algún número aparece en la parte izquierda, pero no en la derecha, quiere decir que el nodo o arco se borra cuando se aplica la regla. Si el número aparece en la parte derecha, pero no en la izquierda, el nodo o arco se crea al aplicar la regla. Los nodos y arcos de la parte izquierda pueden tener etiquetas especiales de cara a la especificación de las condiciones de correspondencia ("*matching conditions*"). Por ejemplo, los atributos de los nodos o arcos de la parte izquierda pueden tener cierto valor, lo que quiere decir que en el modelo deben tener exactamente esos valores para que se produzca una correspondencia. Los nodos o arcos de la regla también pueden tener la marca <*ANY*>, que quiere decir que cualquier valor del atributo va a producir una correspondencia. Con respecto a los atributos de los nodos de las partes derechas de las reglas, se puede incluir cierto valor, lo cual quiere decir que el nodo tomará ese valor al producirse la sustitución. O bien puede tener la marca <*COPIED*> que quiere decir que el valor del atributo se copiará de su homónimo de la parte izquierda de la regla (nótese que aquí es necesario por tanto que el nodo de la parte derecha exista en la parte izquierda). Por último, un atributo puede tener la marca <*SPECIFIED*> (como el atributo *Tcreat* del nodo 7), que quiere decir que se incluye una expresión (en AToM^3 estas expresiones se especifican en Python) que calcula el valor del atributo.

La segunda regla de la gramática también se encarga de generar transacciones, pero no es necesario que la cola no esté vacía para su aplicación. De hecho, como se ha colocado después de la regla 1 – que necesita que la cola no esté vacía – la cola en este caso va a estar vacía, ya que si se aplica esta regla, quiere decir que la regla 1 no ha podido aplicarse. En la figura no se han incluido otras condiciones de aplicabilidad (por simplicidad), como por ejemplo que un generador no puede rebasar un número máximo de transacciones (si éste se ha especificado en el atributo *MaxTransactions*).

La figura 5 muestra las reglas 3 y 4, que se encargan de que los procesos en estado *Idle* consuman transacciones.

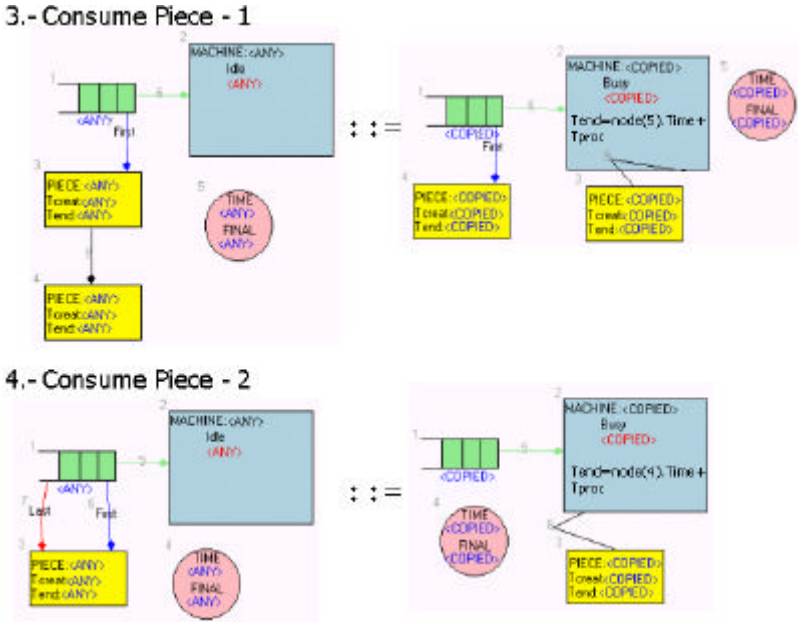

Figura 5: Reglas 3 y 4 de la Gramática para la Simulación: Consumo de Transacciones por Procesos.

Como en la generación de transacciones, tenemos dos versiones de la regla. La primera es aplicable si la cola de entrada al proceso no se queda vacía cuando se consume la transacción. Si esto es así, entonces la primera transacción de la cola es consumida por el proceso, éste pasa al estado "*Busy*", y el atributo *Tend*, que señala el instante en que el proceso terminará, se calcula convenientemente (tiempo actual más tiempo de proceso). La regla 4 es muy similar a la anterior, pero es un caso especial, en el que la cola se queda vacía, con lo cual hay que borrar los enlaces a la primera transacción y a la última (etiquetados *First* y *Last* en la parte izquierda de la regla).

La figura 6 muestra las reglas 5 y 6, que sirven para especificar la terminación de un proceso y por tanto la salida de la transacción a la cola de salida.

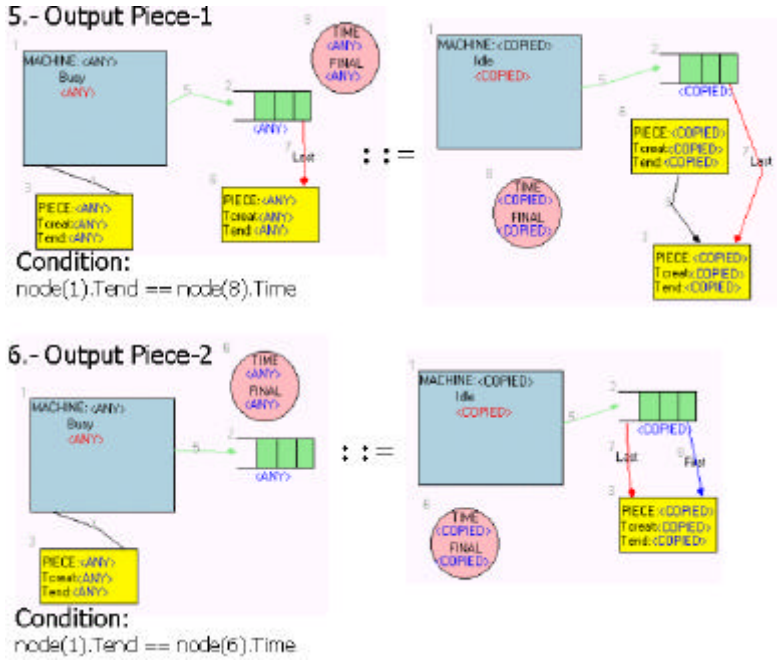

Figura 6: Reglas 5 y 6 de la Gramática de la Simulación: Terminación de Procesos.

Como en los casos anteriores, hay dos versiones de las reglas. La primera se aplica si la cola de salida no está vacía (es decir, el arco *Last* de la cola apunta a una transacción). Para la aplicación de la regla 6 esto no es necesario, y como la regla 5 se comprueba siempre antes, la cola estará vacía cuando se aplique la regla 6. Ambas reglas tienen como condición de aplicación que el tiempo almacenado en el atributo *Tend* del proceso (que es el tiempo en que se programó que el proceso acabaría) sea igual al tiempo actual de la simulación.

Finalmente, la regla 7 se encarga de actualizar el tiempo de la simulación.

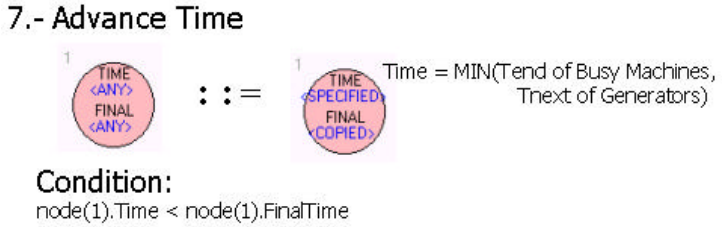

Figura 7: Regla 7 de la Gramática: Incremento del Tiempo de la Simulación.

Esta regla se aplica si ninguna de las anteriores se ha podido aplicar, y simplementa actualiza el tiempo actual de la simulación (atributo *Time* del *Timer*) con el tiempo del próximo evento. Este es el tiempo mínimo de terminación de los procesos y de la generación de transacciones por los generadores. Nótese que la condición de aplicabilidad es que el tiempo actual sea menor que el tiempo final. Si esta condición no se cumple, la regla no se puede aplicar, lo cual quiere decir que ninguna regla se ha podido aplicar (ya que si estamos evaluando esta regla, es que se han evaluado todas las anteriores y no han podido ser aplicadas) y le ejecución de la gramática (y por tanto la simulación) termina.

La siguiente figura muestra algunos momentos en la ejecución de la gramática sobre un modelo formado por dos procesos (máquinas) organizados de manera secuencual y conectados por colas.

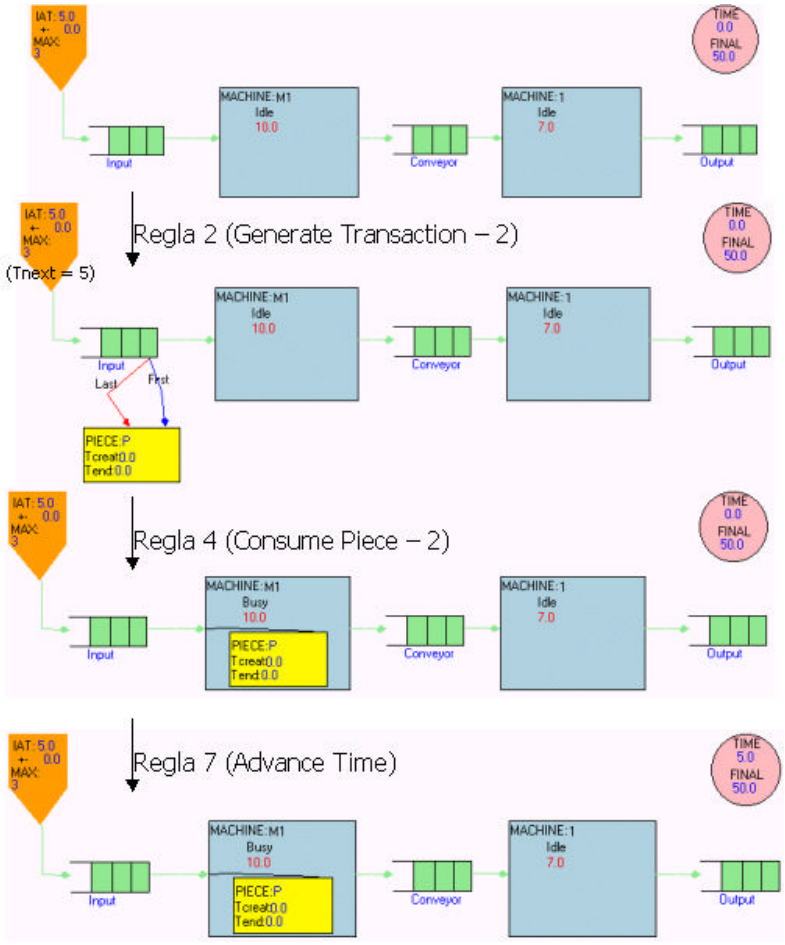

Figura 8: Simulación de un Modelo Mediante la Gramática (3 Primeros Pasos, un Avance de Tiempo)

Durante la simulación, puede darse la situación de que una regla pueda ser aplicable en más de una zona del modelo. Por ejemplo, si las dos máquinas están libres y tienen una pieza en la cola de entrada. Entonces el módulo de reescritura de grafos de AToM^3 ofrece varias posibilidades. La primera es preguntar al usuario en qué zona del grafo desea aplicar la regla (en el ejemplo, tendría que indicar cuál de las dos máquinas debe consumir primero la pieza). La segunda posibilidad que ofrece AToM^3 es que la herramienta elija una zona al azar. Finalmente, si las zonas en que se puede aplicar la regla son disjuntas, se puede especificar que AToM^3 aplique la regla en las dos zonas a la vez (es decir, en el ejemplo, ambas máquinas consumirían las piezas a la vez). Este no determinismo daría igual en la presente simulación, ya que se pueden aplicar varias reglas de la gramática en el mismo intervalo de tiempo (es decir, antes de ejecutar la regla 7 que avanza el tiempo), con lo cual el resultado de la simulación sería el mismo.

La ejecución de la gramática se puede realizar paso a paso (es decir, el usuario presiona en un botón para continuar tras cada ejecución de una regla), o bien de forma animada. En AToM^3, se puede establecer cuánto tiempo tiene que pasar desde la ejecución de una regla hasta que se ejecuta la siguiente. En el ejemplo esto es muy útil, ya que la animación puede realizarse en tiempo real. Por ejemplo, si la regla 7 avanza el timer en 5 unidades de tiempo, la simulación tardaría 5 segundos en ejecutar la regla 7.

6. Trabajo Relacionado

Hay otras herramientas similares en la comunidad de gramáticas de grafos, como GenGed [Bard02], que es una herramienta para construir entornos visuales "*dirigidos por la sintaxis*". Las ideas de la herramienta son similares a las de AToM^3, con la excepción de que en AToM^3 no se utiliza un enfoque "*dirigido por la sintaxis*" en los entornos de modelado que genera. En algunos casos, un entorno "*dirigido por la sintaxis*" es díficil de usar y restrictivo para el usuario final. En GenGed la organización de la apariencia gráfica del modelo se expresa como un problema de satisfacción de restricciones. En AToM^3 el diseñador del meta-modelo debe cuidar la presentación gráfica por medio de pre- y post- condiciones, expresadas en Python (y que se evalúan cuando se producen eventos, al estilo de *Visual Basic*). A veces, esto puede resultar más complicado que usar restricciones, pero otras veces nuestro enfoque ofrece más flexibilidad.

Otras herramientas, como DiaGen [Mina02] (y AToM^3), pueden combinar características de edición "*free-hand*" y "*dirigido por la sintaxis*". DiaGen es una herramienta basada en gramáticas de hipergrafos. El usuario debe especificar el lenguaje visual que quiere definir de forma textual y obtiene una colección de clases Java que se complementan mediante otra librería Java. En AToM^3, la especificación del lenguaje visual (el meta-modelo) se hace de forma gráfica, y los fichero que genera AToM^3 se cargan de nuevo en la herramienta para obtener el entorno de modelado. No hay diferencia estructural entre los editores generados por AToM^3 y los editores que los generaron (de hecho, los editores generados, se pueden utilizar para generar otros [deLa02]). Una de las mayores diferencias con otras herramientas es que en AToM^3 todo es un modelo, o ha sido definido mediante un modelo, y por tanto, el usuario puede cambiarlo.

Hay herramientas de meta-modelado que se utilizan en la comunidad de simulación, como por ejemplo DoME [DOME99] [Pere02], aunque las manipulaciones de los modelos deben expresarse en lenguajes textuales. En el caso de DoME, las computaciones se expresan en el lenguaje Smalltalk o en Alter (similar a Lisp).

7. Conclusiones y Trabajo Futuro

En este artículo se ha presentado AToM^3, una herramienta de meta-modelado que permite la generación rápida de entornos de modelado especializados. Las capacidades de las herramientas de modelado generadas se pueden ampliar mediante la especificación de gramáticas de grafos para la manipulación de modelos. Como ejemplo, se ha mostrado el meta-modelado de un formalismo visual para la simulación discreta así como una gramática de grafos para su simulación, con un enfoque educativo. Según el conocimiento del autor, es la primera vez que se ha utilizado una gramática de grafos para la especificación de un simulador en el que la base temporal son los números reales con este enfoque.

Las ventajas del uso de una herramienta como AToM^3 para la generación de herramientas de modelado son claras: la productividad se aumenta notablemente. Se puede generar una herramienta para formalismos como el presentado en este artículo en muy pocas horas. Además, se pueden especificar computaciones sobre los modelos sin usar algoritmos textuales (mediante gramáticas de grafos), y sin apenas conocimiento de la estructura interna de AToM^3.

La herramienta se ha usado con éxito en varios cursos de doctorado (en la Universidad Autónoma de Madrid) y licenciatura (en la Escuela de Informática de la Universidad de McGill). No sólo los alumnos pueden experimentar con formalismos y gramáticas ya construidas, sino que ellos mismos pueden construir formalismos mediante meta-modelado. Este proceso de meta-modelado ayuda a conocer todos los detalles del formalismo que se describe y por tanto mejora mucho su concocimiento sobre éstos. Además la

especificación gráfica y formal de manipulaciones sobre los modelos también hace que los alumnos obtengan un mejor comprensión del proceso de la transformación.

Los conceptos presentados en este artículo no sólo son útiles en el contexto de la simulación, sino que son extrapolables a otras áreas. Por ejemplo, estas técnicas pueden ser útiles para mostrar conceptos relacionados con estructuras de datos (mostrando operaciones y algoritmos sobre diversos tipos de listas, pilas, grafos en general, etc.), esta es una de las posibles líneas futuras de trabajo. En el futuro también se seguirá trabajando tanto en mejorar el lenguaje de simulación presentado, como en el meta-modelado de otros formalismos. Asimismo, se seguirá mejorando la propia herramienta AToM^3, por ejemplo añadiendo el concepto de "*herencia*" al meta-formalismo DER que se usa para definir otros formalismos.

Agradecimientos

Este trabajo ha sido parcialmente subvencionado por el Ministerio Español de Ciencia y Tecnología, con el proyecto número TEL1999-0181.

8. Referencias

[ATOM03] Página web de la herramienta AToM^3: `http://atom3.cs.mcgill.ca`

[Bard02] Bardohl, R., 2002. "*A Visual Environment for Visual Languages*". Science of Computer Programming 44, pp.: 181-203.

[Blon96] Blonstein, D., Fahmy, H., Grbavec, A.. 1996. "*Issues in the Practical Use of Graph Rewriting*". Lecture Notes in Computer Science, Vol. 1073, Springer-Verlag, pp.38-55.

[Booc99] Booch G., Rumbaugh J., Jacobson I. 1999. "*The Unified Modeling Language User Guide*". Addison Wesley.

[deLa02] Juan de Lara, Hans Vangheluwe. 2002. "*AToM^3: A Tool for Multi-Formalism Modelling and Meta-Modelling*". En Fundamental Approaches to Software Engineering FASE'02, dentro del congreso European Conferences on Theory And Practice of Software Engineering. ETAPS'02. Grenoble, Abril 2002. Lecture Notes in Computer Science 2306, pp.: 174-188. Springer-Verlag.

[deLa02b] Juan de Lara, Hans Vangheluwe. 2002. "*Computer Aided Multi-Paradigm Modelling to process Petri-Nets and Statecharts*". Lecture Notes in Computer Science 2505. pp.: 239-253.

[DOME99] Guia de DOME: `http://www.htc.honeywell.com/dome/`, Honeywell Technology Center. Honeywell, 1999, version 5.2.1

[Dörr95] Dörr, H. 1995. "*Efficient Graph Rewriting and its implementation*". Lecture Notes in Computer Science, 922. Springer.

[Ehri99] Ehrig, H., Engels, G., Kreowski, H.-J., and Rozenberg, G. 1999. "Handbook of Graph Grammars and Computing by Graph Transformation". Vol.2: Applications, Languages, and Tools. World Scientific.

[Gord96] Gordon, G. 1996. "*System Simulation*", Prentice Hall. Second Edition.

[Gray00] Gray J., Bapty T., Neema S. 2000. "*Aspectifying Constraints in Model-Integrated Computing*", OOPSLA 2000: Workshop on Advanced Separation of Concerns, Minneapolis, MN, October, 2000.

[Mina02] Minas, M. 2002. "*Concepts and Realization of a Diagram Editor Generator Based on Hypergraph Transformation*". Science of Computer Programming 44, pp.: 157-180.

[Pere02] Pereira Remelhe, M., Engel, S., Otter, M., Derarade, A., Mosterman, P. 2002. "*An Environment for Integrated Modelling of Systems with Complex Continuous and Discrete Dynamics*". In Lecture Notes in Control and Information Systems. Vol(279) pp.:83-105.